|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  **Project I**  **Đề bài:**  **Tìm hiểu tóm tắt về Qt Framework**   |  |  |  | | --- | --- | --- | | Sinh viên thực hiện | MSSV | Lớp | | Nguyễn Đức Đạt | 20172458 | ĐTVT-08.K62 | | Vũ Đức Thái | 20172804 | ĐTVT-08.K62 | | Bùi Đức Dũng | 20172492 | ĐTVT-08.K62 |     Giảng viên hướng dẫn: PSG.TS Nguyễn Đức Minh    Hà Nội, 4-2020 |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  **Project I**  **Đề bài:**  **Tìm hiểu tóm tắt về Qt Framework**   |  |  |  | | --- | --- | --- | | Sinh viên thực hiện | MSSV | Lớp | | Nguyễn Đức Đạt | 20172458 | ĐTVT-08.K62 | | Vũ Đức Thái | 20172804 | ĐTVT-08.K62 | | Bùi Đức Dũng | 20172492 | ĐTVT-08.K62 |     Giảng viên hướng dẫn: PSG.TS Nguyễn Đức Minh    Hà Nội, 4-2020 |

MỤC LỤC

[Lời nói đầu 3](#_Toc39317108)

[PHẦN 1 Cấu trúc class của Qt Framework 4](#_Toc39317109)

[1.1 Mô hình phân cấp của Qt class 4](#_Toc39317110)

[1.2 Parenting system 5](#_Toc39317111)

[1.3 Signals and slots 6](#_Toc39317112)

[1.3.1 The Observer pattern 6](#_Toc39317113)

[1.3.2 Tổng quan “Signals and slots” 7](#_Toc39317114)

[1.3.3 Đặc điểm của signals và slots 8](#_Toc39317115)

[1.3.4 The Meta Object (siêu đối tượng) 9](#_Toc39317116)

[PHẦN 2 Mô tả hoạt động của một chương trình Qt cơ bản 10](#_Toc39317117)

[2.1 Tạo một project Qt mới 10](#_Toc39317118)

[2.2 Hoạt động của một chương trình Qt đơn giản 11](#_Toc39317119)

[2.3 Cơ chế biên dịch của một chương trình Qt 13](#_Toc39317120)

[Kết luận 14](#_Toc39317121)

[TÀI LIỆU THAM KHẢO 15](#_Toc39317122)

DANH MỤC HÌNH ẢNH

[Hình 1.1: Cấu trúc phân cấp của class Qt 3](#_Toc39239959)

[Hình 1.2: UML class diagram for observer pattern 5](#_Toc39239960)

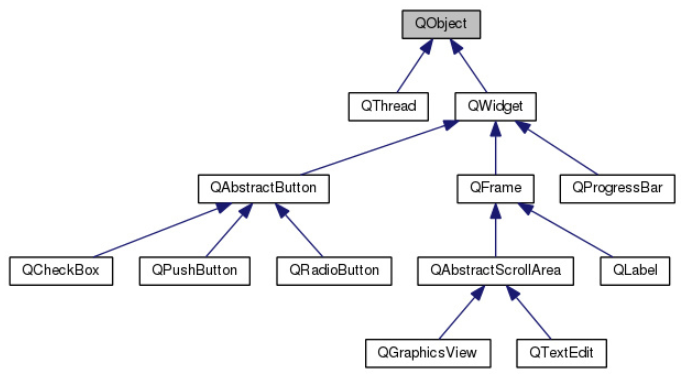
[Hình 2.1: Biên dịch chương trình đơn giản “Hello world!” 11](#_Toc39239961)

# Lời nói đầu

Ngôn ngữ lập trình C++ là một trong những ngôn ngữ lập trình hướng đối tượng mạnh và phổ biến hiện nay do tính mềm dẻo và đa năng của nó. Không chỉ các ứng dụng được viết trên C++ mà cả những chương trình hệ thống lớn đều được viết hầu hết trên C++. C++ là ngôn ngữ lập trình được viết trên nền tảng của C, không những khắc phục được một số đặc điểm của ngôn ngữ C mà quan trọng hơn, C++ cung cấp cho người sử dụng một phương tiện lập trình theo kỹ thuật mới: lập trình hướng đối tượng. Đây là kỹ thuật lập trình được sử dụng hầu hết trong các ngôn ngữ mạnh hiện nay, đặc biệt là các ngôn ngữ hoạt động trong môi trường Windows như Microsoft Access, Visual Basic, Visual Foxpro… Xuất phát từ thực tiễn này em sẽ khái quát lại một số những hiểu biết về các kiến thức liên quan đến lập trình hướng đối tượng. Em xin chân thành cảm ơn thầy đã tạo điều kiện và cung cấp những tài liệu cần thiết trong quá trình hoàn thành bài báo cáo này. Bài tiểu luận không tránh khỏi những thiếu sót. Em mong nhận được ý kiến đóng góp của thầy để bài báo cáo được hoàn thiện hơn nữa.

# Cấu trúc class của Qt Framework

## Mô hình phân cấp của Qt class

Qt sử dụng một cách phổ biến tính kế thừa, đặc biệt là trong Widgets moldule. Đồ thị dưới đây thể hiện sự kế thừa của các lớp trong Qt class:

Hình 1.1: Cấu trúc phân cấp của class Qt

QObject là một lớp cơ bản nhất trong Qt framework . Phần lớn các lớp trong Qt đều được kết thừa từ class này. QObject cung cấp những khả năng rất mạnh mẽ như:

* Tên đối tượng (object name): bạn có thể thiết lập một tên, như là 1 chuỗi (string) trong một đối tượng và tìm chúng bằng tên.
* Hệ thống kế thừa (parenting system): Mô tả những quan hệ trong các lớp kế thừa (cụ thể ở phần sau)
* Signals and slots: là hai khái niệm mức cao được Qt cung cấp dựa trên Observer Pattern (sẽ được đề cập ở phần sau)
* Quản lý các hoạt động (event management)

Widgets có khả năng đáp ứng các sự kiện, sử dụng hệ thông kế thừa (parenting system) và cơ chế “signals and slots”. Mọi Widgets đều kế thừa từ QObject. Widget cơ bản nhất là lớp Qwidget, nó chứa hầu hết các thuộc tích được sử dụng để mô tả các cửa sổ (window) hoặc các thuộc tính như vị trí và kích thước, con trỏ chuột, các chú thích…

*Nhận xét*: Trong Qt, một widget có thể cũng là một cửa sổ. Gần như tất cả các thành phần đồ họa đều được kế thừa từ class QWidget, ví dụ như một số lớp:

* QAbstractButton (Là một lớp cơ sở của tất cả các button)

QPushButton

QCheckBox

QRadioButton

* QFrame (Là lớp tạo khung)
* QLable (Là lớp tạo ra text hoặc ảnh)
* Sự kế thừa được thực hiện để tạo ra các tiện ích quản lý tài nguyên. Các thuộc tính được chia sẻ như kích thước và vị trí có thể được sử dụng trong các thành phần đồ họa khác, và lớp QAbstractButton cung cấp những thuộc tính cơ bản để chia sẻ tài nguyên tất cả các button.

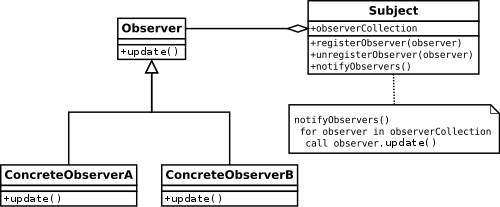
## Parenting system

Parenting system là một cách thuận tiện để giải quyết các vấn đề với các đối tượng trong Qt framework, đặc biệt là với Widget. Bất kỳ đối tượng nào kế thừa từ lớp QObject đều có thể là lớp mẹ hoặc lớp con. Hệ thống phân cấp như vậy có một số đặc điểm sau:

* Khi một đối tượng bị hủy, tất cả các lớp con của nó cũng bị hủy theo. Vậy nên, việc gọi delete trở thành lựa chọn trong một số trường hợp chắc chắn.
* Tất cả các các lớp QObject đều có phương thức *findChild* và *findChildren,* chúng có thể được sử dụng để tìm kiếm lớp con thuộc đối tượng đã đưa ra.
* Các Child Widgets trong lớp QWidget sẽ tự động xuất hiện bên trong Parent Widget.

## Signals and slots

### The Observer pattern

* Hầu như tất cả các UI toolkits đều có một cơ chết dò hành động của người dùng, và phản hồi lại hành động đó. Một vài trong số chúng sử dụng *callbacks.* Nhưng ở mức độ cơ bản, tất cả toolkits đều lấy ý tưởng từ *observer pattern.*
* Observer pattern được sử dụng khi một đối tượng có khả năng quan sát muốn thông báo đến những đối tượng khác về sự thay đổi trạng thái. Một vài ví dụ cụ thể cho khái niệm như sau:
* Người dùng kích đúp chuột vào một button và một menu được hiện lên
* Một trang Web sau khi tải xong, sẽ tiến hành trích xuất những thông tin từ trang trang web này.
* Người sử dụng duyệt danh sách các vật phẩm (trong một app bán hàng), khi tiếp cận để cuối, những vật phẩm khác sẽ được tải thêm
* Observer pattern được sử dụng ở mọi nơi trong GUI applications và thường thì nó dẫn đến việc sử dụng một số mã soạn sẵn (boilerplate code). Qt đưa ra cơ chế “signals and slots” với ý tưởng loại bỏ những mã soạn sẵn này, cung cấp cho chúng ta một cú pháp rõ ràng và không dư thừa.
* UML class diagram:

Hình 1.2: UML class diagram for observer pattern

### Tổng quan “Signals and slots”

* Signals and slots: Thay vì sử dụng observer pattern thì Qt cung cấp hai khái niệm mức cao dựa trên Observer Pattern. Có nhiệm vụ giám sát và thông báo trong các trạng thái:
* Một Signal là một thông báo được một đối tượng gửi, trong phần lớn trường hợp dùng để thông báo các thay đổi trạng thái
* Một Slot là một hàm, nó được sử dụng để truy cập và phản hồi đến Signal
* Ví dụ thực tế trong lớp QPushButton như sau:
* Clicked
* Pressed
* Released

Khá rõ ràng, Những signals được gửi đi khi người dùng click, press hoặc release vào button, ở đây có một vài slots từ các lớp khác nhau: QApplication::quit, QWidget::setEnabled, QPushButton::setText. Do đó để phản hồi một signal, một slot phải được kết nối với một signal. Qt cung cấp phương thức QObject::connect để kết nối giữa signal và slot

* Về cơ bản, signals và slots là các phương thức (có thể có đối số hoặc không có đối số) không có giá trị trả về. Khái niệm signal được định nghĩa như một phương thức không chính thức, và slot thực sự được nhắc đến như một phương thức, nó có thể được gọi như các phương thức thông thường hoặc trong phản hồi đến signal
* *Transmitting information:*
* Cơ chế “*Signals and slots”* ngoài việc hữu ích để phản hồi hành động với các button, thì nó có thể được sử dụng để kết nối thông tin, chúng ta lấy ví dụ như trong khi chúng ta phát một bản nhạc, chúng ta cần biết còn bao nhiêu khoảng thời gian nữa trước khi bài hát kết thúc. Một trình phát nhạc có thể có một class để kiểm tra quy trình xử lý này (cụ thể thì class sẽ gửi đi tín hiệu theo chu kỳ, với giá trị tương ứng), signal này có thể được kết nối với QProgressBar (được sử dụng để hiển thị tiến trình)

void MediaProgressManager::tick(int miliseconds);

* Và để chúng ta biết được thông tin quy trình sử lý, chúng ta sẽ sử dụng một slot trong class QProgressBar

void QProgressBar::setValue(int value);

* Chúng ta có thể thấy các tham số đầu vào cùng kiểu dữ liệu (int) , đặc biệt là kiểu trả về là void (không trả về giá trị nào). Ở bước này chúng ta kết nối signal và slot không có cùng kiểu tham số đầu vào thì ta sẽ nhận được một cảnh báo khi biên dịch. Đó là vì signal truyền thông tin sử dụng cùng một tham số với các slot, tham số đầu tiên của signal được truyền đến đầu vào của slot đầu tiên, thứ hai,…

QObject::connect: Incompatible sender/receiver arguments

* Đoạn code kết nối sẽ giống như sau:

MediaProgressManager \*manager = **new** MediaProgressManager();

QProgressBar \*progress = **new** QProgressBar(window);

QObject::connect(manager, SIGNAL (tick(int)), progress, SLOT (setValue(int)));

### Đặc điểm của signals và slots

* Một signal có thể được kết nối với nhiều slot
* Nhiều signal có thể được kết nối với một slot
* Một signal có thể được kết nối với một signal (chuyển tiếp). signal thứ hai được gửi nếu signal đầu tiên được gửi.

### The Meta Object (siêu đối tượng)

* Qt đưa ra một hệ thống siêu đối tượng (nghĩa đen là đối tượng bao trùm) là một hướng tiếp cận của một vài mô hình có thể lập trình mà trong C++ thuần túy không có, một số đặc điểm sau:
* Sự nội quan: cung cấp khả năng kiểm tra một loại nào đó trong thời gian biên dịch.
* Các mệnh hàm không đồng bộ.
* Để sử dụng khai thác khả năng của siêu đối tượng, người ta có thể sử dụng một lớp con của class QObject và chỉ định nó như một siêu đối tượng trong trình biên dịch. Mã được tạo ra sẽ chứa các đặc trưng “signals and slots”, phương thức (nó được sử dụng để truy xuất lượng lớn thông tin từ những lớp được đánh dấu, xử lý tài nguyên…) Tất cả những thông tin này được truy cập thông qua phương thức sau:

**const** QMetaObject \* QObject::metaObject () **const**

* Class QMetaObject chứa tất cả những phương thức để xử lý siêu đối tượng

# Mô tả hoạt động của một chương trình Qt cơ bản

## Tạo một project Qt mới

* Ban đầu ta sẽ thực hiện với một dự án trống (empty project), chúng ta tiến hành: File > New file or Project > Orther Projects > Empty Qt Project
* Sau đó chọn project folder và đặt tên, phiên bản Qt sử dụng trong dự án
* Trong project file (.pro). Qt sử dụng công cụ dòng lệnh ( command line tool) để phân tích những file trong project, do đó tạo ra “makefile”. Các file được sử dụng để biên dịch và xây dựng ứng dụng. Công cụ này được gọi là *qmake* (chúng ta không cần thiết quan tâm đến qmake, bởi vì Qt Creator sẽ thực hiện công việc đó cho chúng ta)
* Trong phần project file này, chúng ta thực hiện một và dòng code ngắn gọn luôn luôn được viết ở đầu mỗi project, sau đây là một ví dụ khái báo đơn giản:

TEMPLATE = app

TARGET = name\_of\_the\_app

QT = core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

* *TEMPLATE* mô tả kiểu của chương trình, nó có thể là một ứng dụng, một thư viện, hoặc đơn giản chỉ là một thư mục con.
* *TARGET* là tên của ứng dụng hoặc thư viện.
* *QT* khai báo các thư viện (Qt Moldules) được sử dụng trong project. Như ở ví dụ trên, chúng ta cần hai thư viện đó là QtCore và QtGUI.
* *GreaterThan (…)* sẽ khai báo phiên bản Qt sử dụng trong chương trình và thêm Widgets vào trong QT
* Khi ta thực hiện thêm các “C++ Source File” hoặc các “C++ Header File”, một dòng mới sẽ tự động thêm vào *file \**.*pro*

HEADERS += first\_file.h second\_file.h

SOURCES += first\_file.cpp second\_file.cpp

## Hoạt động của một chương trình Qt đơn giản

Ta thực hiện đoạn code đơn giản sau trong *main*.*cpp*:

#include *<QApplication>*

int main(int argc, char \*\*argv)

{

QApplication app (argc, argv);

**return** app.exec();

}

* Ban đầu ta khai báo thư viện <QApplication>, QApplication là một lớp rất quan trọng, hàm tạo của đối tượng nhận các đối số đầu vào của hàm main. Ngoài ra nó cũng có nhiều thứ khác nữa, đáng chú ý nhất là “event loop”. “event loop” là một vòng lặp chờ người sử dụng đưa input vào ứng dụng GUI. Khi phương thức app.exec() được gọi “event loop” được khởi động.
* Khi biên dịch chương trình, ứng dụng được khởi động và không có trả về. Hiển nhiên là như vậy, bởi vì “event loop” đang được chạy để chờ người dùng đưa ra các thao tác, giống như kích đúp chuột trên GUI…, nhưng ở trường hợp này chúng ta không đưa ra bất kỳ sự kiện nào để chương trình sử lý, nên nó sẽ rơi vào vòng lặp vô hạn.

Chúng ta cùng thử với một vài thao tác tiếp theo trong dòng code sau:

#include *<QApplication>*

#include *<QPushButton>*

int main(int argc, char \*\*argv)

{

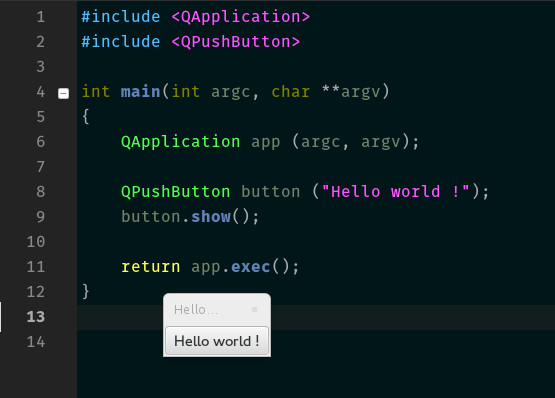
QApplication app (argc, argv);

QPushButton button ("Hello world !");

button.show();

**return** app.exec();

}

Biên dịch code và sau đó chúng ta nhận được :

Hình 2.1: Biên dịch chương trình đơn giản “Hello world!”

## Cơ chế biên dịch của một chương trình Qt

Qt Creator thực hiện việc gọi ra hệ thống biên dịch cho chúng ta

Trong những chương trình đơn giản, dễ dàng biên dịch bằng tay, tạo ra các file đối tượng, sau đó liên kết chúng. Nhưng đối với những dự án lớn hơn, các dòng lệnh trở nên khó viết. *Qmake* đi kèm cùng với Qt và nó tạo ra những *makefile* cho chúng ta (thực tế chúng ta có thể sử dụng những thứ khác, nhưng trong ví dụ này đề cập đến qmake). Với cú pháp đơn giản, nó tạo ra *makefile* – cái mà được sử dụng để biên dịch một chương trình Qt. Nhưng đây không phải là mục đích duy nhất của nó. Qt sử dụng các siêu đối tượng để mở rộng phạm vi chức năng của C++, và *qmake* sẽ có trách nhiệm chuẩn bị các *makefile* (chứa các đoạn trích xuất từ siêu đối tượng này).

Tóm lại, Qt app được biên dịch theo 3 bước sau:

* File \*.pro được Qt Creator đọc để mô tả dự án biên dịch
* Một *makefile*  được tạo ra và sử dụng *qmake*
* Chương trình được biên dịch sử dụng *make (nmake* hoặc *jom* trên windows)

# Kết luận

Lập trình hướng đối tượng được xây dựng dựa trên nền tảng của khái niệm lập trình có cấu trúc và sự trừu tượng hóa dữ liệu. Do đó, hai kĩ thuật lập trình Factory pattern và Finite State Machine pattern là hai kĩ thuật rất quan trọng trong lập trình hướng đối tượng. Đồng thời các kỹ thuật này sẽ làm nền tảng cho lập trình các hệ thống nhúng (embedded system), một kiến thức rất quan trọng với sinh viên ngành điện tử - viễn thông. Qua đó, nhóm em xin gửi lời cảm ơn chân thành đến thầy Nguyễn Đức Minh đã tạo điều kiện và cung cấp các tài liệu cần thiết cho nhóm trong quá trình học tập và nghiên cứu.

# TÀI LIỆU THAM KHẢO

<https://wiki.qt.io/Qt_for_Beginners>

<https://www.qt.io/download?hsCtaTracking=f24f249b-61fb-4dec-9869-50512342f8d9%7Cf3adf380-4740-4f7e-9e49-d06fa99445fa>

<http://www.laptrinhtanbinh.com/course/1/Lap-trinh-voi-ngon-ngu-C/25/3.1.-Gioi-thieu-ve-thu-vien-Qt>